PMX1c Tile Accelerator Specification

Document ID : HTile02

Issue Number : Draft DE

Issue Date : 7th April 1998

Author : Andrew Webber

DOCUMENT HISTORY

Issue	Date	Changes / Comments
Draft A	25th Feb 1998	First Draft.
Draft B	4th Mar 1998	Major change to compensate for the removal of the AGP address LUT (for grid reasons).
Draft C	20th Mar 1998	Minor tweaks, instruction re-groupings, etc.
Draft D	7th April 1998	Updates and changes to the software part of the specification.
Draft E	26th May 1998	Updates and changes to the software part of the specification.

BIBLIOGRAPHY

[1] CLX1 functions and Internal Parameter Format	Pete Leaback	CLXFORM.DOC v1.16 20 October, 1997
[2] PMX1 PowerVR2 3D Data Format	John Metcalfe	PMX3D01F.DOC
[3] PMX1B Tiling Accelerator Specification	Andrew Webber	HTILE01C.DOC

TABLE OF CONTENTS

1. INTRODUCTION

This document outlines the specification of the tile accelerator (TA) on PMX1c. This specification builds on the PMX1b specification discussed in HTile01*.doc. The major differences between the two specifications is that the PMX1c specification includes support for macrotiling and adds the copying of object data into the frame buffer into the TA hardware specification. Otherwise, the inputs data formats remain fundamentally the same (i.e. the input vertex data is supplied in CLX1 native format). Lastly, note that the PMX1c tile accelerator uses a fixed tile size of 32x16 (WxH) with all units such as screen area and clipping rectangles being specified in terms of tiles.

2. THE TILE ACCELERATOR HARDWARE

The block diagram for the PMX1c tile accelerator is shown in Figure 1. The main differences for this arrangement as opposed to that used in PMX1b is that the tile accelerator now has a write port for copying data through while tiling. Some details of the arrangement have been changed (e.g. the TA instruction set and functionality), however, most of these details of these changes are hidden from the host by the software running on the primary core.

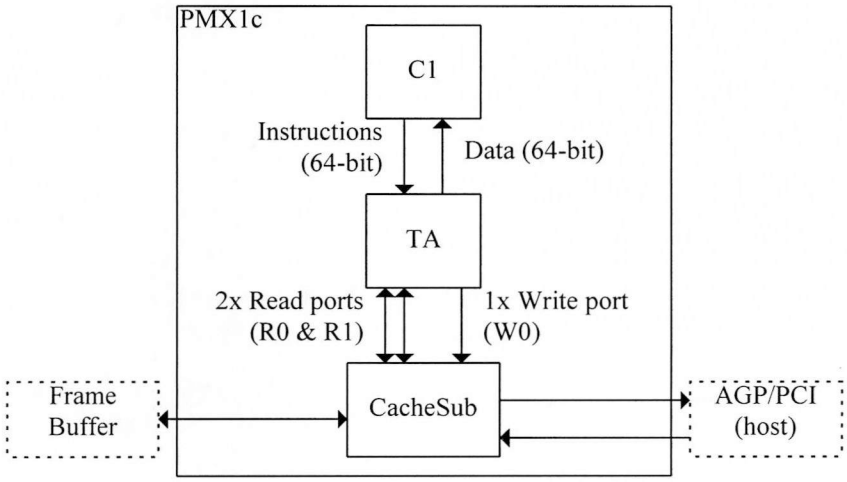


Figure 1

2.1 Tile Accelerator Instructions

To support the new features a new set of TA instructions have been developed that extend upon the functionality of the PMX1b C1/TA interface. These instructions (which are a superset of the two used in PMX1b) are shown in Figure 2 (with instruction 0 being at the top and instruction 3 being at the bottom):

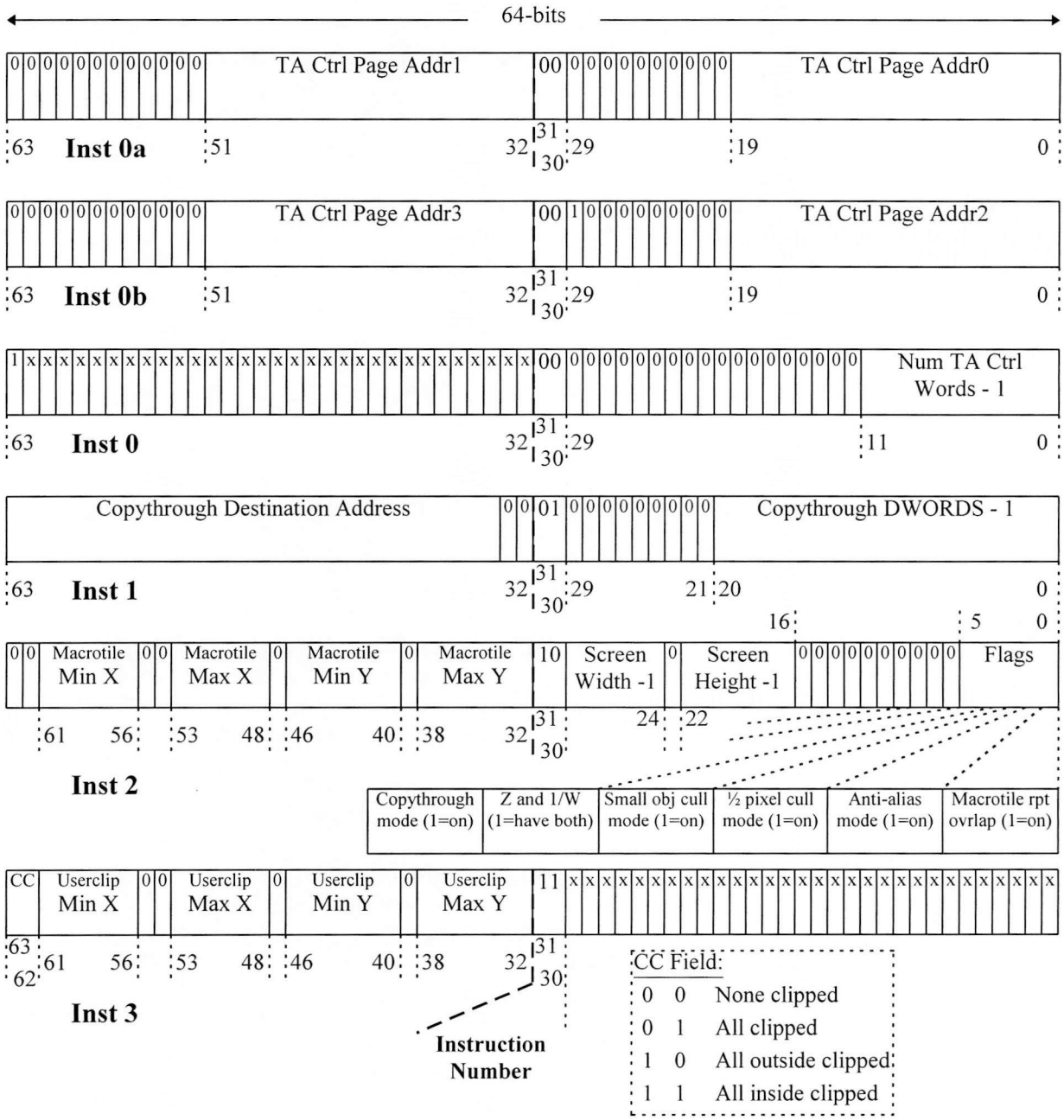


Figure 2

One feature not indicated by Figure 2 is the location of the 'start tiling' command. In fact, this is implied by sending instruction 0 with the number of control words and top bit of the 64-bit instruction set to '1' (3rd instruction from top). Also of note is the fact that for a scene made up of a number of groups of objects (which may have a different set of parameter types, etc. - e.g. opaque/translucent) only registers 0 and 3 are likely to need to be set up differently for the different groups.

The input data (TA ctrl list) must be arranged into a set of at most four physical pages, the addresses of which must be fed into the tile accelerator before the tiling operation is started.

This setup has been specified as a pair of sub-groups of instruction 0 (top two instructions on the figure).

When performing copythrough the object pointers need to be regenerated using a process that is compatible with the render hardware. The arrangement used by the render hardware and the process used to obtain the output object pointer address when copying data through the tile accelerator is shown in Figure 3:

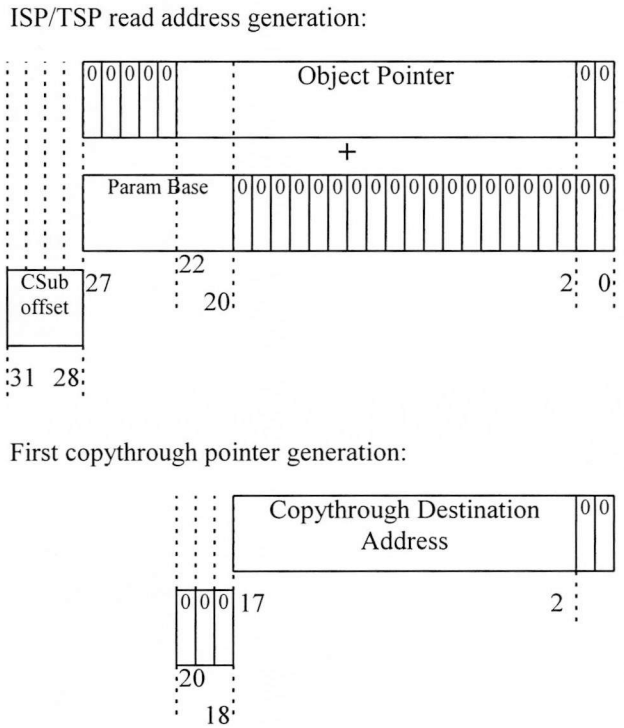


Figure 3

Figure 3 shows that the first copythrough object pointer should have its top three bits set to zero and use bits 17 down to 2 of the copythrough destination address (NEEDS TO BE SPECIFIED). Successive pointers should increment from this start position and increments should also affect the top bit three bits. This gives an available range of 0Mb + CopythroughOffset through to 8Mb - 1 (i.e. a total range of 8Mb - OffsetIntoFirstMb).

2.2 Tile Accelerator Control List Data Format

The tile accelerator control list contains all of the object pointers for the objects that are to be tiled. In addition, in copythrough mode this list contains optional ISP/TSP instructions that are copied with the vertex data into the output. If we are not running in copythrough mode (i.e. similar mode to that used in PMX1b) the ISP/TSP instructions must be interleaved with the vertex data that is referenced by the object pointers. This arrangement is shown in Figure 4:

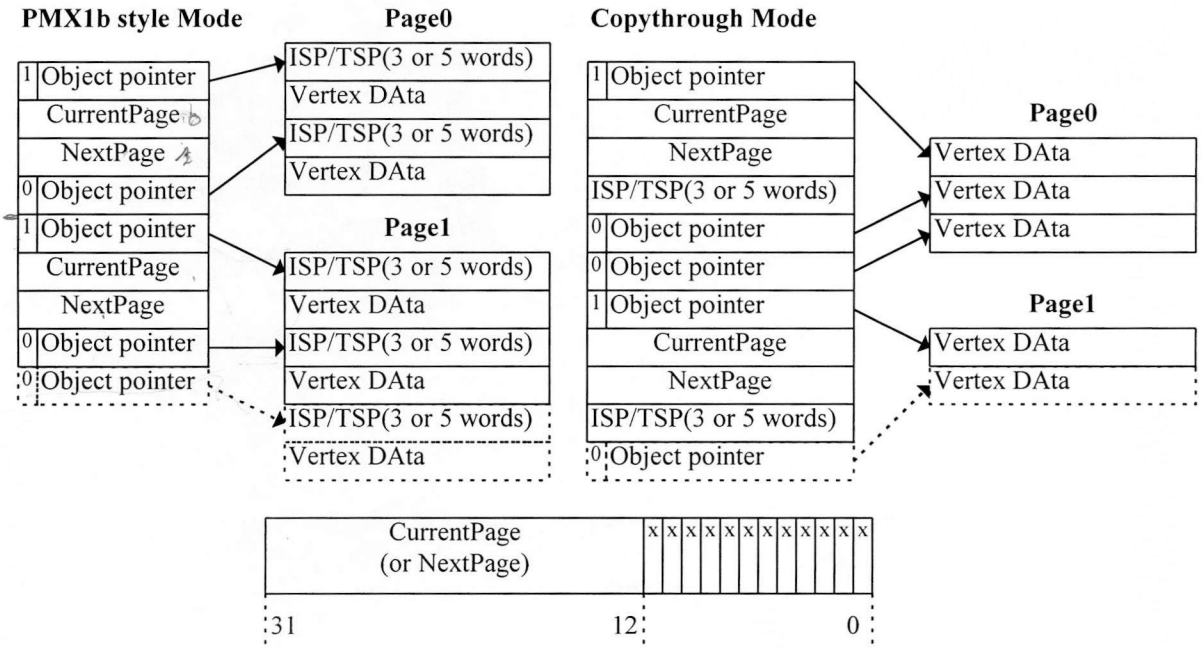


Figure 4

The object pointers should be CLX1 native strips in the format shown below:

Bits 31	*30-25*						*24*	*23-21*	*20-0*
X	Mask						Shadow	Skip	Triangle Strip Start (32bit Word Address)
	T0	T1	T2	T3	T4	T5			

From PMX1c onwards it is proposed that the strip mask field may contain a partial set of flags (e.g. T1 and T3 set all others not). This allows some other types (such as fans) to be represented by adding in extra vertices that do not warrant a triangle of their own. A result of this change is that the tile accelerator must now find the rightmost triangle flag and use this to work out the number of vertices used for this object. Thus, if T0 is triangle 1 and T5 is triangle 6 there will (TriangleNumber + 2) vertices for a given strip. If any mask flag is set in the input object pointer it may be set in the output object pointer (depending upon bounding box evaluation), otherwise it must remain clear in the output object pointer.

If no mask bits are set (i.e. the strip is empty) the number of vertices for this object are unspecified. In this situation the tile accelerator should ignore the strip completely (with the exception of control word/page information updates indicated by the top bit). The tile accelerator should also not copy any data through for this object. This behaviour is possible because the tile accelerator evaluates a complete physical address for each distinct object and can therefore skip empty ones without needing to know the size.

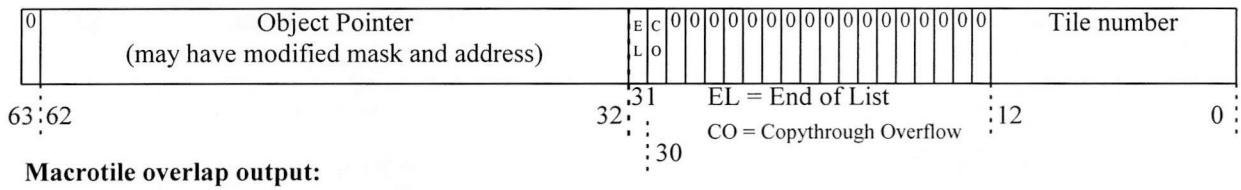
Note that the 0's and 1's shown in Figure 4 refer to the top bit of the object pointer (which otherwise retains the same format as PMX1b and CLX). After processing any flags added into this list will be lost allowing the rendering hardware to interpret these pointers correctly. To avoid the need for a translation LUT object pointers are tagged to indicate that a new pair of addresses for this page and the next one should be read in. In copythrough mode this bit also accompanies changes in ISP/TSP instruction state and should therefore should be set if

either a new page needs to be set or a new ISP/TSP instruction group needs to be set. In non-copythrough mode this bit only indicates the presence of page information. In both cases the new page information should first be used with the object to which the flag is attached (as shown in Figure 4). CurrentPage and NextPage should be 32-bit addresses of which only the top 20-bits are used. In principle, CurrentPage and NextPage should be set with the first object in a given page. The NextPage field is needed to allow the last object on a page to overrun into another page (making the host software simpler and more memory efficient and the tile accelerator hardware more complicated).

2.3 Tile Accelerator Output Data

The tile accelerator outputs two types of data. The first of these is the combined ISP/TSP instruction and vertex data which is written out (to the framebuffer) by the tile accelerator when running in copythrough mode. The second type of data output by the tile accelerator consists of commands sent back to the primary core indicating the data structure updates required to correctly tile the given set of object data. Having initiated the tile accelerator the appropriate thread on the primary core will slavishly follow this data stream until explicitly told to stop by the tile accelerator. The format of the primary core control data is shown in Figure 5:

'Normal' output:



Macrotile overlap output:

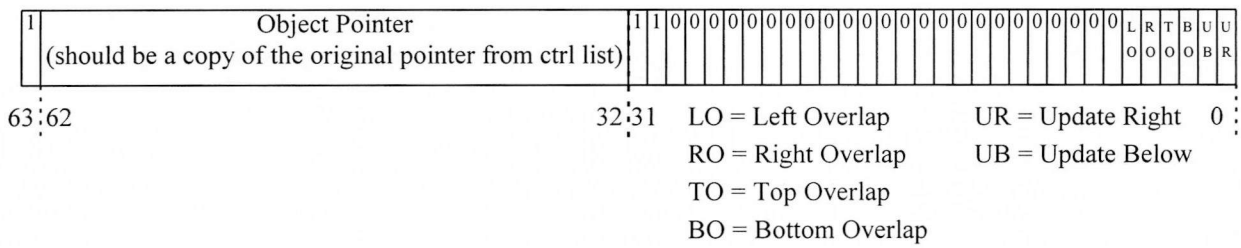


Figure 5

For normal outputs the format is pretty much as it was in PMX1b. Note, however, the addition of the copythrough overflow flag. This bit will only be set with the 'End of List' being set at the same time (to allow handling to be taken out of the main program flow). When a copythrough overflow occurs the tile accelerator will clear its read ports but not write any more data or send the primary core any more objects. In this situation the primary core will need to restart the tile accelerator (having negotiated for a larger copythrough buffer. The tile number generated by the tile accelerator is calculated by the following equation:

Tilenum = (TileXPos-MacroTileMinX)+((TileYPos-MacroTileMinY)*(MacroTileMaxX+1-MacroTileMinX)).

This is because only a subset of the whole screens' tiles will actually be used for the macrotile (as shown in Figure 6 for a 4x4 macrotile at 4, 4). Thus, the tile accelerator should return tile numbers within the macrotile (as shown by the M0-M15 numbers in the figure).

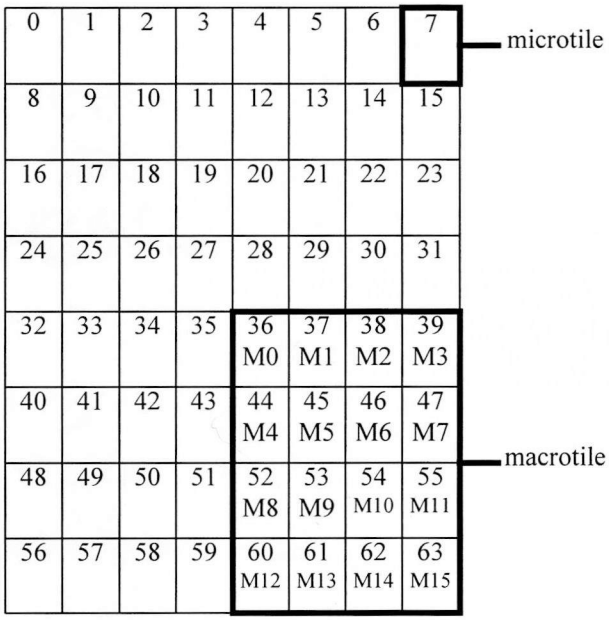


Figure 6

Macrotile overlap outputs indicate that an object overlaps another macrotile. In this situation the input lists for the affected macrotiles must be altered to include this object. It is assumed that we will always macrotile from top left to bottom right (and therefore only need to update macrotiles to the right or directly below this one). From the set of information provided by the tile accelerator the only bits that are really relevant are the 'UR' and 'UB' bits which indicate where overlap should be added. Note, that either or both of these bits may be set and when both are set the overlap is only written to the right macrotile and the below macrotile.

When overlap occurs the original object pointer that caused the overlap needs to be copied into the appropriate neighbouring macrotiles. In addition, the CurrentPage/NextPage and ISP/TSP control words associated with this object pointer will need to be copied and so the tile accelerator will feed these through directly after the overlap notification. Either 3 or 5 ISP/TSP control words may be sent depending upon the value of the 'shadow' field with normal object pointer outputs following directly after these extra control words.

The control words, sent from the tile accelerator to the primary core for overlapping macrotiles, are sent via the top 32-bits of the tile accelerator's 64-bit output stream. While the tile accelerator is outputting the overlapping objects control words the bottom 32-bits of the tile accelerators output have no defined value or meaning. Finally, note that, as with the copythrough overflow special case described above, the macrotile overlap effectively indicates 'End of List' to allow the handling software to be extracted from the main loop (however, in this case the list is not actually terminated).

Macrotiling should only report overlap for macrotile boundaries that fall within the screen area. For example, if an object overlaps the right hand side of the macrotile in the example arrangement shown above no overlap should be generated as the object also overlaps the screen edge.

3. PRIMARY CORE SOFTWARE

The primary core software handles the interaction with the host and manages the rest of the tiling function. In essence, the primary core takes a command from the host and parcels it up for the TA hardware block. Thereafter, the software will handle the construction of the data structures (region array and associated object lists) until told to stop by the TA co-processor (via the 'End of List' flag).

If the tile accelerator is to tile a whole scene (without macrotiling) a single primary core control block is used - the screen effectively consists of a single macrotile. However, when a the macrotile size is smaller than the screen size a linked list of macrotiles must be provided by the host software, as shown, for example in Figure 7 (for the scene/macrotile arrangement shown in Figure 6):

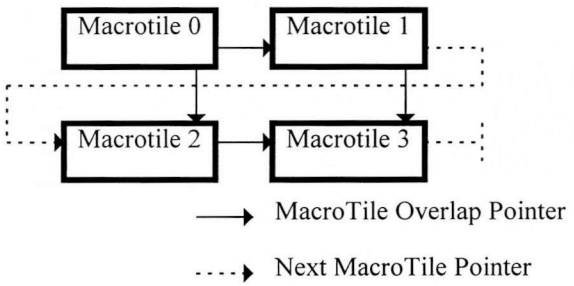


Figure 7

The links to the right and below macrotiles allow the primary core software to handle macrotile overlap conditions by inserting data into other macrotiles. Note that the right hand side of macrotile 1 links to macrotile 2 even though there should be no overlap for the right hand side of this tile (as it is the same as the right hand side of the screen). This link is needed to allow the primary core to process all the macrotiles and should never be used for macrotile overlap. Based upon the specification, shown in Figure 2, macrotiles are arbitrary rectangles described in terms of microtile poisitions.

All macrotiles adjacent to the bottom of the screen should have a NULL (0) below reference pointer and the bottom-right corner macrotile should also have a NULL right reference pointer. The primary core will process the list of macrotiles by using the right reference pointer of the current macrotile to find the next macrotile (until a NULL pointer is found). It is recommended that the right-most and bottom-most macrotiles are clipped to the screen edge (although this may be found to be unnecessary). Given this strategy the primary core will always process the screen are in left to right scan lines of macrotiles from top to bottom until the bottom right corner is reached.

3.1 Data Structures

A macrotile is described by the following data structure (where a full scene can be pictured as a single macrotile):

*1 ** Macrotile Block	*N ** Pass/List Block

In this structure, the macrotile block has the following form (aligned to an 8-byte boundary):

DWORD TA Inst 2 (low DWORD) *(see Figure 2 for details)*
DWORD TA Inst 2 (high DWORD) *(see Figure 2 for details)*
DWORD Right Macrotile Reference Pointer
DWORD Below Macrotile Reference Pointer
DWORD Log2NumPasses
DWORD MacroTileW*MacroTileH*4

The C1 memory buffer is the area of (usually frame buffer) memory that the primary core uses to build the region arrays and pointer lists to hold the tiled data. Each region may hold any of a fixed group of four lists (opaque, opaque modifier, translucent and translucent modifier). In addition to this, in PMX1c the concept of multiple passes has been added to allow some special rendering tricks to be employed. Multiple passes require multiple regions, which must follow one another in memory for a given tile as follows (for 4 tiles and 4 passes):

Tile0, Pass0	Tile0,Pass1	Tile0, Pass2	Tile0, Pass3
Tile1, Pass0	Tile1,Pass1	Tile1, Pass2	Tile1, Pass3
Tile2, Pass0	Tile2,Pass1	Tile2, Pass2	Tile2, Pass3
Tile3, Pass0	Tile3,Pass1	Tile3, Pass2	Tile3, Pass3

To keep the primary core code simple when dealing with multiple passes the number of passes must be a power of two (hence the Log2NumPasses in the data structure above). In addition, the primary core uses some of its memory buffer area for pointer list tail pointers while running the tiling (otherwise known as temporary tail pointers).

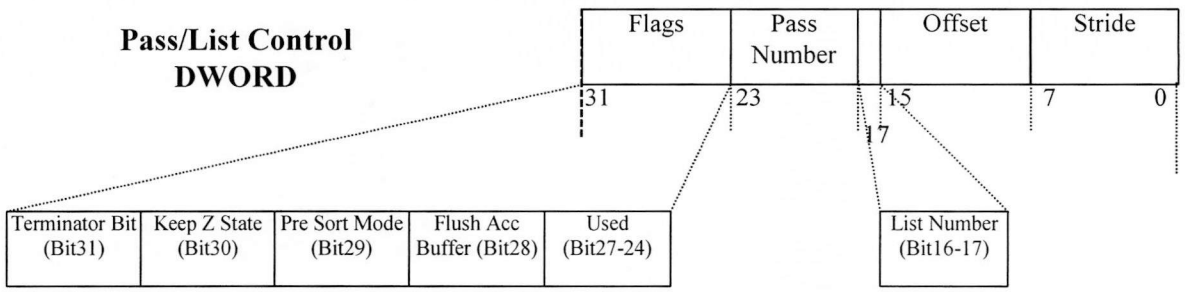


Figure 8

Lastly, the pass/list block has the following form (aligned to an 8-byte boundary):

DWORD Pass/List Control DWORD *(see Figure 8 for details)*
DWORD TA Inst 3 (high DWORD) *(see Figure 2 for details)*

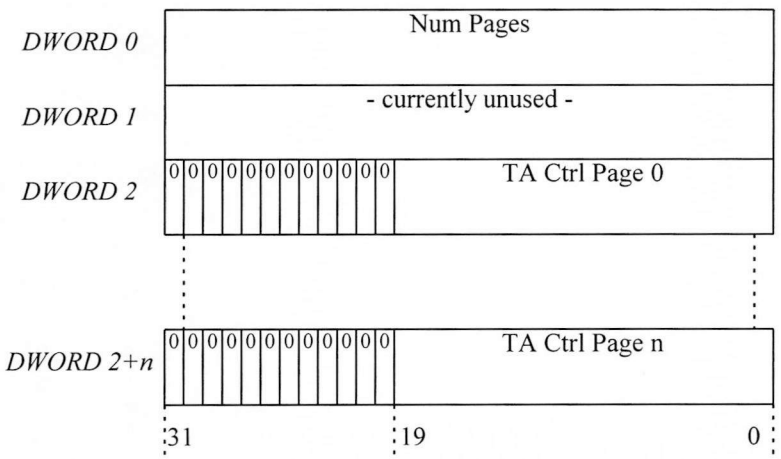
DWORD Num TA Ctrl List DWORDS
DWORD TA Ctrl List Page Table Pointer

When multiple macrotiles are used to describe a scene the same number of passes and lists must be used for each macrotile (to allow for handling of macrotile overlap). Any unused passes or lists for a given scene should have a ctrl list DWORD count of 0. Any lists or passes that are not used in any macrotile may be ommitted from the pass/list array. The pass/list array should follow on (contiguously in memory) from the macrotile block.

The pass/list array is terminated with the top-bit in the first word set to '1' (i.e. Figure 8) in the last pass/list block. This is because the number of pass/list blocks depends not only upon the number of passes but also upon the lists used within those passes.

The pass/list control DWORD also defines information such as the pass and list number of the current pass/list block. The stride and offset values are used to interleave translucent objects (List 2) for trilinear and multi texturing. If the list is not translucent then the offset should be set to 0 and the stride to 1. Also included are region header flags.

The TA Ctrl list page table is a pointer to an array of physical page address that will be sent to the tile accelerator so it can read the control list (see Inst 0 in Figure 2 for details). The first entry in the page table should be a count of the number of pages in the table (and should be aligned on an 8-byte boundary). The second word is currently an empty placeholder (to keep the entries suitably aligned). The page numbers themselves should just be the top 20-bits of the physical address of the used pages, as shown below:



When macrotiling using several macrotiles it is recommended that the page table is initialized with an extra page to allow macrotile overlap data to be inserted. This overlap requirement applies to every macrotile with the top-left macrotile being the only exception as overlap can only be to the right or below of the current macrotile (and they are processed in a fixed top-left to bottom-right order).

3.2 Memory Map

For the each scene as a whole the memory layout shown in Figure 8 would be used (where this example is for the copythrough case):

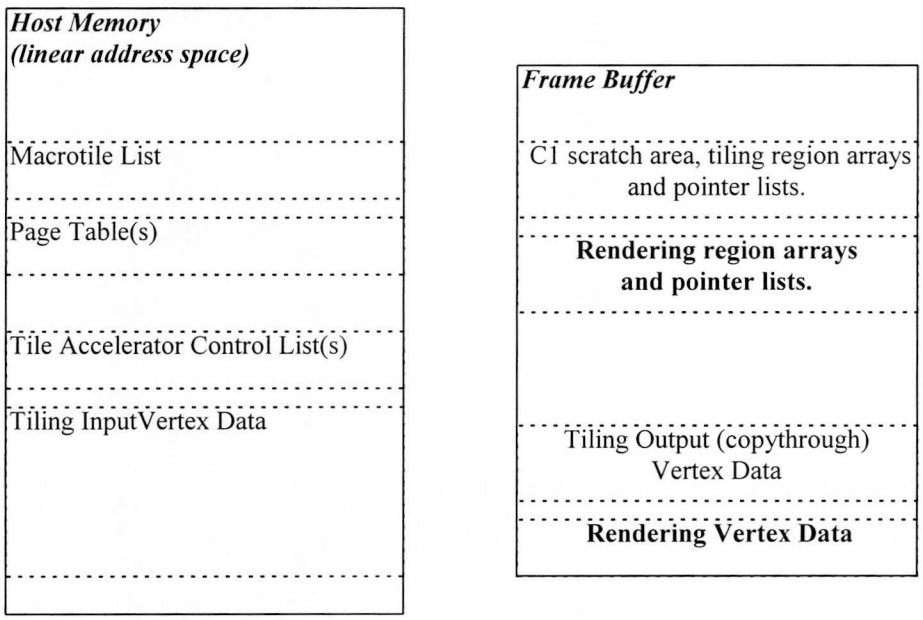


Figure 8

As shown in Figure 8 it is assumed that most of a scenes control data (before tiling) will come from host memory. This figure shows the host memory as a linear memory space, however, in reality the various blocks of data will be formed from sets of physical pages (4096 bytes) which are unlikely to be contiguously allocated. This specification assumes that a scenes macrotile list will be contained within a single page and that groups of tile accelerator control words will be allocated in groups of pages (starting on a page boundary). It is also essential that page tables do not cross page boundaries. In fact, the only exception to this rule is the input vertex data, which can be spread across page boundaries (as long as a given strip is in a maximum of two physical pages). Figure 8, also highlights the double-buffering needed to allow a macrotile/scene to be rendered while the next is being tiled. It is reasonably likely that the tiling and rendering will swap between two common buffers on a macrotile by macrotile basis.